TMA4106 Matematikk 2 Oblig

1)

Kode:

```
import numpy as np

h = 0.1

def f(x):
    return np.e**x

def f_derivert(x):
    return (f(x+h)-f(x))/h
for i in range(9):
   print(f_derivert(1.5))
   print(h)
   print(f(1.5)-f_derivert(1.5))
   h = h/10
```

Output:

Dette viser at når h blir mindre enn 1.0x10^-9, så blir metoden mer unøyaktig.

```
1e-05
-2.2408571680010425e-05
4.4816913105094605
1.0000000000000002e-06
-2.240171395939683e-06
4.481689295232626
1.0000000000000002e-07
-2.248945616400988e-07
4.481689064306237
1.0000000000000002e-08
6.031827481933376e-09
4.481689686031131
1.0000000000000003e-09
-6.156930663081539e-07
```

2)

Kode:

```python
import numpy as np

h = 0.1

def f(x):
    return np.e**x

def f_derivert(x):
    return (f(x+h)-f(x-h))/(2*h)
for i in range(7):
   print(h)
   print(f_derivert(1.5))
   print(f(1.5)-f_derivert(1.5))
   h = h/10
```

Output:

```
1e-05
4.481689070434669
-9.660450217552352e-11
1.0000000000000002e-06
4.4816890696353076
7.027569637330089e-10
1.0000000000000002e-07
4.481689073188021
-2.849956715067492e-09
```

Her vil metoden bli mindre presis når h = 1 x 10^-6. Dette kan forklares ved å bruke taylorsrekker. Restleddet for denne metoden kan skrives som:

$$\frac{f'''(x)}{3!}h^2 + \frac{f'''''(x)}{5!}h^4$$

Her ser man at restleddet vil bli veldig lite selv for større verdier av h, som gjør at denne metoden blir raskere nøyaktig enn metoden over.

3)

Kode:

```
import numpy as np

h = 0.1

def f(x):
    return np.e**x

def f_derivert(x):
    return (f(x-2*h)-8*f(x-h)+8*f(x+h)-f(x+2*h))/(12 * h)
for i in range(9):
   print(h)
   print(f_derivert(1.5))
   print(f(1.5)-f_derivert(1.5))
   h = h/10
```

Output:

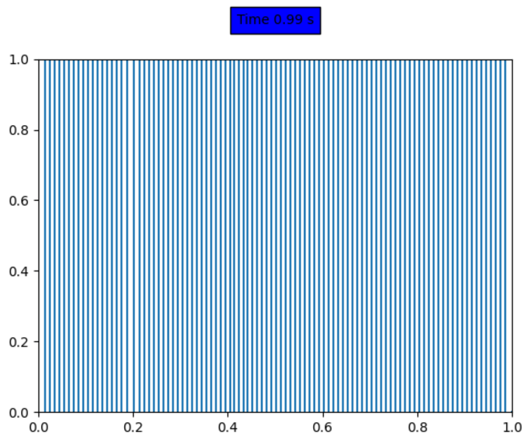
```
0.1
4.481674113579644
1.4956758420225924e-05
0.01
4.481689068844186
1.4938787984419832e-09
0.001
4.481689070337191
8.730793865652231e-13
0.0001
4.48168907034215
-4.085620730620576e-12
1e-05
```

Her vises det at metoden blir mer unøyaktig når h = 0.0001. Restleddet for denne metoden ble:

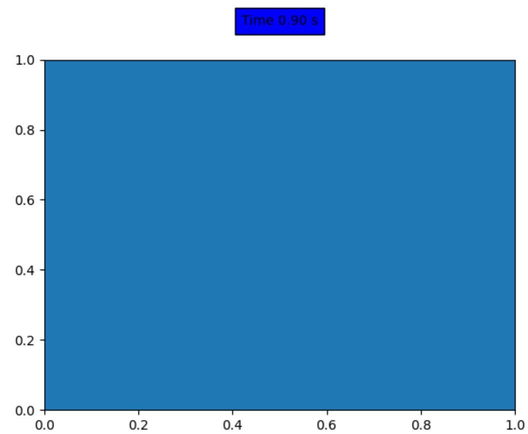
$$-\frac{1}{30} f^5(x) h^4$$

Her vil restleddet bli enda mindre for mindre verdier av h, som betyr at metoden blir en god approksimasjon selv for større verdier av h.

4)
For eksplisitt ble det vist at metoden er veldig sensitiv, som gjorde at ved for eksempel h = k = 0.01, så ble plottet seende ut som i figur 1 under. Når h = 0.1 og k = 0.001, så får vi på starten en sinus funksjon som går beveger seg mot null når t = 1. Dette stemmer overens med randkrav og initialbetingelser. Hvis derimot k > h, som for eksempel h = 0.001 og k = 0.1, så får vi en funksjon som i figur 2 under. Eksplisitt Euler er altså en veldig ustabil metode.



Figur 1: h = k = 0.01



Figur 2: h = 0.001, k = 0.1.

5)
Ved implementering av implisitt Euler, ble det sett at dette er en mye mer stabil metode som gir fornuftige resultater for mange flere verdier av h og k. Ved å bruke h = k = 0.01, blir det en funksjon som går raskt mot null siden h er så stor. Implisitt Euler vil altså funke for mange verdier av h og k og derfor være mer stabil.

6)
Ved implementering av Crank-Nicholson ble det vist at dette er en mer stabil metoden en eksplisitt, men mindre stabil enn Implisitt Euler. Når h = k er metoden mer stabil ved små verdier, men metoden vil allikevel fornuftige resultater i forhold til Eksplisitt Euler. I likhet med de to andre metodene vil fungerte Crank-Nicholson også for h >>k. Crank-Nicholson fungerte også for k > h, men var da ganske ustabil og gav ikke en fin funksjon som for implisitt Euler. For samme verdier av h og k, ble det altså vist at Implisitt Euler er den mest stabile metoden, Crank-Nicholson er litt mindre stabil, og Eksplisitt Euler er en veldig sensitiv metode.